# Model-based diagnosis for avionics systems using minimal cuts

**Fabien Kuntz** [1,2], **Stéphanie Gaudan** [1], **Christian Sannino** [1], **Éric Laurent** [1], **Alain Griffault** [2], **Gérald Point** [2]

[1] THALES *Avionics S.A., F-31036 Toulouse, France*
{*fabien.kuntz, stephanie.gaudan, christian.sannino, eric.laurent*}*@fr.thalesgroup.com*
[2] LABRI-*CNRS, Université Bordeaux 1, F-33405 Talence, France*
{*fabien.kuntz, alain.griffault, gerald.point*}*@labri.fr*

## ABSTRACT

Increasing complexity of avionics systems leads to reconsider methods that are used today to diagnose system failures. In addition, the size of avionics systems is an important characteristic to consider and that narrows the possible methods we can use to diagnose our systems. Therefore we need to work both on the mastery of the avionics system size and on the improvement of actual diagnostic techniques.

In parallel, problems of diagnosis and safety analysis for avionics context have a lot of common points. Here, we use safety analysis experience to set up a diagnostic method and try to get it close to methods used in safety analysis.

This paper presents how we use model-based reasoning, rule-based reasoning and safety particular techniques to set up an automated diagnosis solution. It gives a feedback on the use of minimal cuts to diagnose large industrial systems.

## 1 CONTEXT

Over the past few years, avionics systems complexity has grown strongly. Even quite recently, each avionics function was implemented on only one hardware unit. In these conditions, detecting and locating a failure was quite easy and describing the relations between observations and failures was feasible by hand. But today, softwares are present everywhere. Furthermore, the way in which hardware supports software has evolved too. Functions are supported by several hardware units and one hardware unit can be shared by several functions. It is common to use *Avionics Full DupleX (AFDX)* networks or *Integrated Modular Avionics (IMA)* in avionics systems, which are good examples of the way avionics systems evolve towards resources sharing. Here are the main reasons why systems become more complex and harder to analyse.

Today the most practical technique to make diagnosis for avionics maintenance is to use rule-based reasoning. Diagnosis rules represent the relations between observations that are thrown by the monitoring system and their possible causes. These rules are mainly handmade based on human expertise. The increasing level of system complexity makes this operation very difficult. Indeed, to find all the causes that could have thrown a monitoring observation is not an activity designed for humans. Such an exhaustive operation is a task more adapted to computers.

Another important point to consider is that our work takes place in an industrial context. That brings several constraints. We must find a solution which scales to sizes of real systems and that can be integrated into our current engineering processes.

The application of model-based techniques in an industrial context is a well-identified problem. In (Console and Dressier, 1999), the authors explained the situation: model-based diagnosis is a primary need for industry, but it needs efforts from both "application peoples" and academic researchers. Today, notably thanks to workshops like "Workshop on Principles of Diagnosis", application side is more considered. But the effort that requires modelling of industrial systems is often too important for companies that prefer to keep going with their current processes. These reasons motivate us to find a method that is generic, automatic, and which takes into account these fears.

In this context, we aim at defining a diagnosis approach allowing us to master both the systems' growing complexity and their size. In the following section we detail further our problem and compare diagnosis and safety analysis. In the section 3 are exposed our choices and solutions, and also methods and process of our diagnostic chain. This is completed by a brief presentation of features of a case study on which we apply this method. Finally our conclusion integrates some hints for future work.

## 2 PRESENTATION OF THE PROBLEM

Several definitions of diagnosis can be found in the literature (Hélouët *et al.*, 2006). The *Oxford Dictionary* official definition presents diagnosis as *"the identification of the nature of an illness or other problem by examination of the symptoms"*. In the systems diagnosis field, it means to *explain the observed situation*. For a given diagnosis problem, i.e. given a situation, solving it means finding its causes.

In this section we describe the type of systems that we deal with and present the diagnostic problem that we want to solve. We briefly give the problems of safety analysis and show how they overlap with diagnostic ones. Then we describe techniques used to analyze systems in the safety assessment domain, and we give an idea of how we can use these techniques to solve our diagnostic problems.

### 2.1 Avionics systems and diagnostic problem

Systems that we want to diagnose are avionics systems, composed of several types of components. Broadly speaking, we can identify two types of items: physical (hardware) components and functional components. Functional components must fulfil a precise purpose and physical components enable functional ones to accomplish their work.

Physical components are, for example, computation units used for calculation tasks, I/O components allowing I/O functions to send and receive messages and data, network components (switches, cables, ports, ...) dispatching messages and data, or also probes measuring for instance pressure or temperature. Functional components are for example computing functions, I/O functions being in charge of process sending and reception of messages and data, comparison functions comparing several values, or also monitoring functions whose role is to detect, for instance, when data flows, functions, or I/O mechanisms, do not have a correct behaviour. These monitoring functions are primordial for diagnosis because they provide observations which are the starting point of diagnosis. These monitoring observations correspond to the result of the sending of a message from a monitoring function.

We use diagnosis to perform the maintenance of an avionics system. Indeed physical and functional components have failure modes associated to them. They can break down totally, partially, or change their functioning mode, resulting in an unexpected behaviour of the system. To repair the system, the goal of the maintenance action is to repair faulty components, i.e. components that are responsible for the faulty behaviour of the system. The problem of avionics systems diagnosis for maintenance is, given a set of observations provided by the monitoring functions (what call a *situation* in the sequel), to find what are the components responsible for system faulty behaviour.

### 2.2 Parallel with safety analysis

Avionics safety is related to the mastering of critical events. In simple words, the mission of safety engineers is to ensure that the occurrence of a critical event is very improbable. Thus, the need behind a safety analysis is first to identify all the root causes potentially leading to a critical situation and then use these information to quantify the probability that critical events occur.

We can compare the safety need described here and diagnosis need. Indeed the main need for diagnosis is to identify the causes that can have led a system in a problematic situation. Safety considers only critical situations while diagnosis must be exhaustive, but they both need to know the possible causes. For diagnosis we need to handle all system events, but in compensation, we do not have to take care of failure occurrence probabilities. An obvious common need to safety and diagnosis is to have a way to formalize relations between system events and observations.

To answer to the problems we previously described, safety has found and set up solutions. A commonly accepted solution is to use *fault tree analysis*. A fault tree is a logic tree emphasizing the causes to effects relations. It describes how a critical event can occur. Then the fault trees are combined and used to quantify the occurrence probability of a critical event.

Today, these fault trees are mainly handmade, described by safety engineers. Here again we can draw a parallel between safety and diagnosis community. Both safety and diagnosis aim at automating the generation of their models (fault trees for safety and diagnostic rules for diagnosis). Indeed with the increasing complexity of avionics systems, creating fault trees (as well as diagnostic rules) is a laborious and hard task, and it is even harder to ensure that they are correct and complete. Moreover, when system architecture is modified, all fault trees impacted by the modification must be rebuilt.

New approaches are studied and start to be used for industrial safety analysis. Among them is a model based approach which consists in formally modelling the system behaviours in a dysfunctional way, and then in generating automatically fault tree analyses with an algorithm based on minimal cuts. An example of the use of minimal cuts for fault tree generation can be found in (Tang and Dugan, 2004). A cut represents a combination of failures sufficient to provoke the critical situation. The first advantage of this approach is that the complexity of the system is handled by the use of high-level formalisms. Indeed with that approach, one describes the behaviour of each different component and how these components interact, and the whole complexity of combined behaviours of the components is handled by the formal semantics of the model. Another advantage is that this approach permits to automate the process. On the other hand, modelling a system is a hard task, more natural for human beings than to find all possible causes of a critical situation, but which needs application and time.

The main difference between safety and diagnosis problems concerns the study perimeter; diagnosis is not only interested in critical events. But we have seen that problems are pretty similar. Thus, an improvement could be the creation of a complete model of the system under diagnosis, the use of algorithms based on minimal cuts in order to obtain diagnostic rules and the combination of the results to make a diagnosis. Note that fault tree analysis techniques have already been proposed in the context of diagnosis, e.g. in (Faure *et al.*, 1999) or (Hurdle *et al.*, 2008).

## 3 DIAGNOSIS SOLUTION

In this section we propose a solution for avionics systems diagnosis which considers usual diagnostic methods, safety methods, as well as industrial constraints.

Several choices have been made to fulfil these constraints. Most of these choices have been motivated by

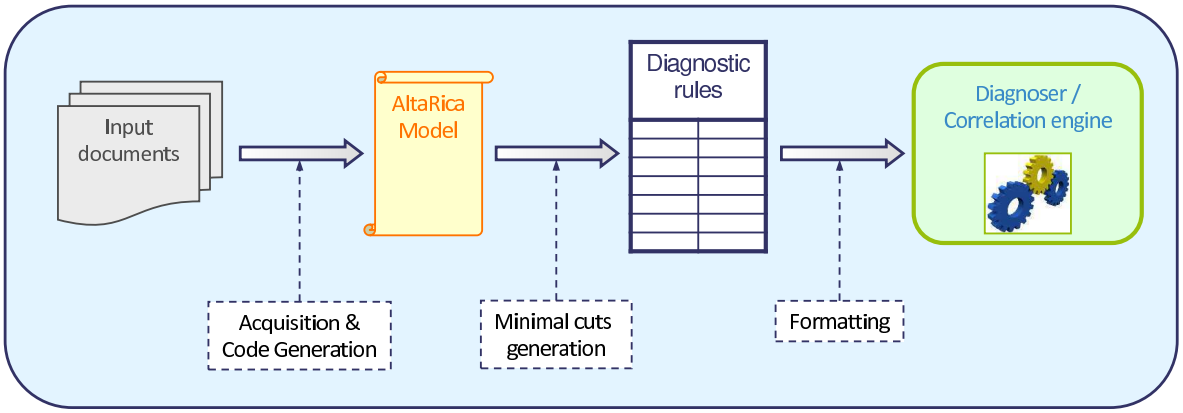


Figure 1: The three phases of the diagnosis chain

our industrial context. Efforts have been made to be compliant with the current maintenance process and to automate the diagnosis workflow. First, we have chosen to model the system under diagnosis with the high-level formalism used by safety engineers. Doing so, we can benefit from safety's modelling work and we can permit safety to benefit from ours, which is important in an industrial context. That choice brings constraints with it: the process of modelling must allow to integrate safety informations into the model. This first choice implies the second one. Indeed we have chosen to generate the model automatically. Then we made the choice to set up a diagnostic method similar to the safety one: we generate diagnostic rules from the diagnostic model thanks to an algorithm producing minimal cuts, and combine these rules to obtain diagnostic results.

Our diagnosis solution is thus split up into three phases depicted on Fig.1:

1. Automatic generation of the model.
2. Automatic generation of the diagnostic rules.
3. Correlation of rules to get the global diagnosis.

In the sequel of that section we present the diagnosis solution and its different phases.

### 3.1 Automatic generation of the model

To explain the first phase of the diagnosis solution, we first introduce the ALTARICA formalism used to model the system. Then we briefly describe the automatic modelling principle and emphasizes some advantages of such an approach.

ALTARICA is a high-level language based on constraint automata (Point and Rauzy, 1999). ALTARICA was created at the end of 90's in the LABRI to answer to industrial needs. This language has been designed to permit the description of both functional and dysfunctional behaviours of critical systems; its formal semantics has been given in (Arnold *et al.*, 1999).

LABRI proposes model-checking tools that support ALTARICA language: ARC with its graphical front-end ALTARICA *Studio* (Griffault *et al.*, 2010) and *MEC 5* (Griffault and Vincent, 2004). There also exists industrial tools based on dialects of ALTARICA, such as *BPA-DAS Safety Designer* of *Dassault Systèmes*, *Cecilia OCAS* of *Dassault Aviation*, *SIMFIA* of *EADS Apsys* or also *RAMSES* of *EADS Airbus*.

We now illustrate ALTARICA language on a small example. In the AltaRica terminology, components are called *nodes*. Fig.2 presents a *node* that models a switch. This node includes a Boolean state variable representing its two possible positions: *on* (closed) and *off* (open). At the initial state, it is *on*. Transitions (keyword `trans`) express that we can modify its state by pushing the switch button through the event $push$. The switch's input and output electrical flows are represented in the model by flow variables $i$ and $o$. The ALTARICA language allows the definition of some invariants called *assertions* (keyword `assert`). Here the unique assertion means: *when the switch is* on*, the flows are equals*. We can see on Fig.3 the labelled transition system representing model's behaviours.

```
node Switch
  state   on :  bool;
  init    on := true;
  flow    i, o :  [0,1];
  event   push;
  trans   true |- push -> on := not on;
  assert  on => (i = o);
edon
```

Figure 2: Model of a switch

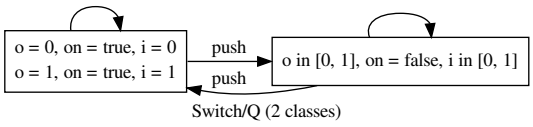


Figure 3: LTS of the switch model

In ALTARICA we also have the notion of hierarchy and events synchronization. We can see on Fig.4 a system composed with two wired switches. The top-level node $System$ is composed of two subnodes of type $Switch$. A constraint, introduced by the keyword `assert`, states that the input of the second switch is wired to the output of the first one. And, finally, an event synchronization, introduced by the keyword `sync`, enforces the `push` events to be simultaneous.

```
node System
  sub     S1,S2 :  Switch;
  assert  S1.o = S2.i;
  sync    <S1.push, S2.push>;
edon
```

Figure 4: Model of a two-switches system

Now we can focus on the method used to automatically generate ALTARICA models.

To have an automatic process of model generation for a system, we must use information we have on the system and define some generation rules.

In addition we want to generalize the process so that we can use it to automatically generate a model for different systems.

In order to be as close as possible to the actual design, we need to take as input data the actual engineering document produced by system designers. There are mainly two kinds of documents:

- Those describing how each components of the system may dysfunction and what are the effects of this dysfunction. We find mainly in this category the FMEA documents (Failure Mode Effect Analysis) required by standard avionics certification procedure such as described in ARP4754A (SAE, 2010).
- Those describing how components are interconnected. We find in this category several types of documents: wiring, functional data flows, network routing, depending on company design process.

To set up the model generation process we keep only information relative to diagnosis and maintenance like the component failure modes, the system faults, the way in which components are interconnected, the different functions of the system in charges of the monitoring, the different messages of maintenance sent by functions, ...

Once the dysfunctional behaviour of components has been collected and once the interconnection between those components has been defined, then we define generation rules to formalize these data into an ALTARICA model. Such rules are:

- A physical component/function will be a node in the ALTARICA model.
- A state of a component will be a state variable of the corresponding node.
- Mode changes are modeled using transitions labelled with failure events of the corresponding node.
- Data and messages will be ow variables in the ALTARICA model.
- ...

With these generation rules, we are able to create an ALTARICA model of the system right in accordance with how engineers has designed it.

Our approach has many advantages. First, we inherit from benefits of a model-based method: once the system is formalized, we can notably perform powerful analyses and model-checking. Moreover with the automatic generation of models, we get around the most important difficulty of a model-based approach which is the creation of the model. There are also other advantages due to our particular industrial context. Actually this method of automatic model generation can be used for safety, one must only point out what are the changes to apply to diagnostic generation rules so that they become safety rules. Furthermore the creation, by hand, of a complete model for each industrial system would be too expensive.

In the following phase we point out how automatic generation of models meets actual processes.

### 3.2 Automatic generation of diagnostic rules

At this step, we have a model which represents the system we want to diagnose, i.e. we have formalized relations between component failure modes and monitoring observations. This is what matters for diagnosis. Now we need to take benefit of these relations to make a diagnosis.

It is necessary for the solution we choose to be integrated into our current processes which are rule-based. As we said in the context section, most of avionics techniques currently used for diagnosis are based on the creation by experts of tables of diagnostic rules. These diagnostic rules, describing what can be the causes for each monitoring observation, can be incomplete since both are handmade and the complexity of systems grows. The idea is thus to automatically create these diagnostic rules from the generated model. This job can be done using minimal cuts of the model.

To explain the second phase of the solution, we first define minimal cuts for a model, and then we explain how we generate diagnostic rules (or minimal cuts) from the model and, what we have not mentioned yet, how we handle the model size issue.

To define what are the minimal cuts we handle, we need to give some other definitions first. Let us start with the definition of a labelled transition system (LTS) that represents the reachability graph of the model.

**Definition 1 (Labelled Transition System)** *A labelled transition system $A = \langle S, I, E, T \rangle$ is a tuple where:*

- *$S$ represents a finite set of states.*
- *$I \subseteq S$ represents the set of initial states.*
- *$E$ is a finite set of events.*
- *$T \subseteq S \times E \times S$ is a set of transitions.*

In the sequel, if $\mathcal{M}$ is an ALTARICA model, then we denote $[\![\mathcal{M}]\!]$ its semantics given as an LTS (Arnold *et al.*, 1999).

LTS are called Finite State Machines (FSM) in (Sampath *et al.*, 1996). Note that in our setting, $T$ is a relation which means that the model is not necessarily deterministic.

In addition we mark some events as *visible*. A visible event is an event which is important to see in the result. For diagnostic problem, it often corresponds to an event representing a failure. For instance, in a model, we can consider failure events and also repair events. On the other hand, identifying repair events in a diagnosis is not mandatory since we only focus on the identification of faulty components. The notion of visibility is different from *observability* encountered in classical frameworks (Sampath *et al.*, 1996) or *controlability* in (Ramadge and Wonham, 1989). Besides, in these frameworks, failure events are considered unobservable and uncontrollable. In our context, visibility attribute only identifies events that must be kept in the result.

In the sequel we denote $E_v \subseteq E$ the set of visible events. We are interested by the computation of *traces* of visible events that leads the system into some states specified as *targets*.

**Definition 2 (Run and trace)** *Given a LTS $A = \langle S, I, E, T\rangle$ and a set of* targets states *$F \subseteq S$, a* run *is a finite sequence $s_0, e_0, \ldots, e_{n-1}, s_n$ such as $s_0 \in I$, $s_n \in F$ and $(s_i, e_i, s_{i+1}) \in T$ for $i \in [0; n-1]$. The sequence $e_0, \ldots, e_{n-1}$ is called the* trace *of the run.*

We denote by $L(A, F)$ the set of traces of $A$ that leads to a state in $F$.

**Definition 3 (Cut)** *If $w = e_0, \ldots, e_{n-1}$ is a sequence of events, the cut associated with $w$ is the set of its visible events.*

$$cut(w) = \{e \in E_v \mid \exists i \in [0; n-1], e = e_i\} \quad (1)$$

Cuts are thus sets of *visible* events. Other events are considered irrelevant for maintenance and are removed from the result.

**Definition 4 (Cuts of a LTS)** *Given a LTS $A = \langle S, I, E, T\rangle$ and a set of targets states $F \subseteq S$, we define the* set of cuts *$Cuts(A, F)$ that lead $A$ into a state of $F$ by:*

$$Cuts(A, F) = \{cut(w) \mid w \in L(A, F)\} \quad (2)$$

Finally we can define the notion of minimal cuts.

**Definition 5 (Minimal cut)** *Given a set of cuts $C$, a cut $c \in C$ is said* minimal *in $C$ iff $\forall c' \in C, c' \subseteq c \implies c = c'$.*

We denote $MinCuts(A, F)$ the set of all the minimal cuts of the LTS $A$ for the targets states $F$.

**Definition 6 (Over-approximation)** *If $C_1$ and $C_2$ are sets of cuts, we say that $C_2$ is an* over-approximation *of $C_1$ iff for any $c_1 \in C_1$ there exists $c_2 \in C_2$ such that $c_1 \subseteq c_2$.*

Minimal cuts are sets of events and not sequences. As a consequence, the logical order of failure occurrences is forgotten. Cuts are thus a coarse and pessimistic abstraction of what actually happens. In the context of maintenance activities, this is not really an issue because all components listed in a cut are checked regardless of the actual sequence.

A similar remark also holds for non observation of some failures. In some systems, it may happen that targets states are reachable only if some components are healthy. Even if healthy components do not appear in minimal cuts they are handled by algorithm used to minimize sets of cuts (Rauzy, 2001). Note that exoneration of healthy components is directly integrated into the model when defining variables used to observe the system.

In our diagnostic problem, we do not exactly handle targets states but rather monitoring observations. But there is obviously a link between them. A monitoring observation is encoded by a variable in our ALTARICA model which is true when the observation is present. A state of the LTS representing the semantics of the model is described by valuations of state and flow variables. We can define targets states for an observation; we call them *observation states*:

**Definition 7 (Observation states)** *Given a LTS $A = \langle S, I, E, T\rangle$ and an observation variable $v$, we define $F_v$, the set of* targets states for *$v$, as*

$$F_v = \{s \in S \mid v \text{ is } true \text{ in } s\} \quad (3)$$

With a LTS $A$, finding the causes of an observation $v$ corresponds to the computation of $MinCuts(A, F_v)$, the minimal cuts of $A$ for the observation states $F_v$ as targets states.

The generated model gives us a LTS from which we now want to compute, for each monitoring observation, the set of minimal cuts. Unfortunately, due to state space explosion, the model can not be used as is.

To tackle this combinatorial explosion, we narrow the model to a submodel which only comprises necessary information to obtain those minimal cuts. More precisely, the algorithm consists of two steps: a first step which considers the global model and computes a dependency graph of the model, and a second step which computes the set of minimal cuts for each observation variable but using a restricted model.

The input of the first step is the ALTARICA model $\mathcal{M}$. Feedback from previously studied systems shows that, despite its size, only a small part of the model impacts the truth value of a given observation variable. Using a static analysis of the model we can reduce it drastically in a conservative way, i.e. the reduced model has the same minimal cuts w.r.t. to a given observation variable.

The static analysis builds a *dependency graph* whose nodes are objects of the ALTARICA model related to assignments of variables: variables, assertions and transitions. In this graph, an edge from one object to another means that the former directly influences the latter. This graph is built as follows:

- A two-way edge is created between an assertion and variables it contains;
- A two-way edge is created between a transition and state variables it assigns a value;
- An edge is created between a variable used in the enabling condition or assigned value of a transition and this latter.

A simple traversal of this graph gives us parts of the model that influences the truth value of an observation variable. In practice, assertions yield lots of two-way

edges. These numerous edges can lead the analysis to build a strongly connected graph containing the whole model and thus we get no gain. To handle this issue we use a pre-processing step that discovers functional dependencies between flow and state variables. These dependencies are then used to remove useless edges.

Once we have a dependency graph of the model $\mathcal{M}$, we can start the second step, which is applied for each observation variable $obs$ of the model. It is composed of five tasks:

1. Computation of the reduced model $\mathcal{M}'$ for $obs$.
2. Encoding of visible events in $\mathcal{M}'$ to obtain $\mathcal{M}''$.
3. Computation of $[\![\mathcal{M}'']\!]$, the LTS of $\mathcal{M}''$.
4. Computation of $Cuts([\![\mathcal{M}'']\!], F_{obs})$.
5. Computation of $MinCuts([\![\mathcal{M}'']\!], F_{obs})$.

Task 1 takes an observation variable $obs$ and first determines the dependency perimeter of $obs$ (i.e. ALTARICA objects). Then it computes the reduced model $\mathcal{M}'$ by projecting $\mathcal{M}$ on this perimeter. We have proved (Griffault *et al.*, 2011) that the reduced model $\mathcal{M}'$ preserves minimal cuts of the original model $\mathcal{M}$: $MinCuts([\![\mathcal{M}]\!], F_{obs}) = MinCuts([\![\mathcal{M}']\!], F_{obs})$.

Task 2 decorates the model $\mathcal{M}'$ with a Boolean variable for each visible event of $\mathcal{M}'$. We obtain the model $\mathcal{M}''$. We have shown that $\mathcal{M}'$ and $\mathcal{M}''$ are bisimilar (Milner, 1980) and thus we have $MinCuts([\![\mathcal{M}']\!], F_{obs}) = MinCuts([\![\mathcal{M}'']\!], F_{obs})$.

Task 3 computes $F_{obs}$. Due to the encoding of $\mathcal{M}''$, $F_{obs}$ is a Decision Diagram (Bryant, 1986) with on one hand states and flow variables included in the dependency perimeter of $obs$, and on the other hand, variables for each visible event that may appear in the result.

Task 4 projects the DD obtained in task 3 on the previously added variables. The new DD we obtain contains only Boolean variables. The remaining variables represent the events that can have led to the observation. We can translate the DD into a Boolean formula to get cuts.

Finally, task 5 computes minimal cuts from cuts using Rauzy's algorithm (Rauzy, 2001).

Our method relies on the particular structure of the generated model. Actually the model contains lots of functional dependencies, what permits to compute submodels little enough to allow computations unfeasible on the global model without combinatorial explosion.

We can also notice that computations on submodels can only be applied to one or few observations. Considering several monitoring observations would increase the probability to get a submodel too large for analysis. Each observation variable would bring with it its own dependencies, and all those dependencies would have to be included in the submodel.

All this process, from reduction of the model to the computation of minimal cuts is implemented in the *ARC* tool (Griffault *et al.*, 2010).

### 3.3 Correlation of rules

At this step, we have met current processes of industrial diagnosis. We have generated the diagnostic rules which are usually written by experts. Nevertheless we must propose a method of correlation for the diagnostic rules which handle the complexity of the rules. Actually generated rules are larger and more complex than the rules written by experts today. The current methods of correlation are designed for smaller rules and can not process complex diagnostic rules we propose here.

We use a method of correlation based on BDDs (Bryant, 1986). We can find an example of the application of BDD-based methods for diagnosis in (Schumann *et al.*, 2004). We give an idea of the different steps of the algorithm of correlation.

An observed situation defines a set of monitoring observations (variables); for each one, we compute a set of minimal cuts. The correlation of these sets yield a new set of cuts that must be, at least, a pessimistic explanation of the situation.

In practice, the correlation is realized as follows. The observed situation gives a set $V$ of monitoring variables. For each such variable $v \in V$, we have computed a set of minimal cuts $C_v$. Each $C_v$ can be interpreted as a Boolean formula $\phi_v$ in disjunctive normal form: each cut forms a conjunctive clause whose variables are failure events and the disjunction of these clauses corresponds to $C_v$. Then we compute the BDD of the conjunction $\bigwedge_{v \in V} \phi_v$. Finally a traversal of the BDD gives a new set of cuts that over-approximates all $C_v$s (in the sense of definition 6). This over-approximation is not an issue because, for maintenance needs, we have to be sure that we identify all the possible causes of a problem.

The last step is to present the results. A simple way to do this, is to order alternatives according to their size, i.e. according to the number of failure events composing them. We present the diagnostic results from the smallest alternatives to the biggest ones. We can even stop the presentation of results as soon as the number of alternatives becomes too big. That presentation of diagnostic results has a particular meaning in avionics context. Indeed it represents single failures, then double ones, ... And this cardinality often matches with probability (i.e., a single failure is often more probable than a double one, ...).

The algorithm we have presented here is simplified. For instance we must be sure that the BDD has the same variables perimeter before doing the logical conjunction. We just wanted to give the main actions and not completely enter in the details.

## 4 CASE STUDY

In this section we give some results and dimensions of an example we have studied.

### 4.1 Case presentation

The case we studied is the avionics suite of a regional jet in the 75 to 95-seats category.

For this study we consider only maintenance information of this avionics suite. Table 1 shortly gives sizes of the system.

### 4.2 Generated model

Table 2 gives sizes of data used for the model generation step: the acquiring of data from engineering input description and the generated model. We remind

Table 1: Sizes opf the system under diagnosis

| # physical components | # physical links | # exchange flows |
|---|---|---|
| 253 | 2772 | 7770 |

that input description can be different data of different types, written by different peoples and for different uses. The given number represents the total of read lines from the different data.

Table 2: Generation files statistics (in number of lines)

| Input description | Generated ALTARICA code |
|---|---|
| 76898 | 35812 |

Let us illustrate the time necessary for modelling generation process for this case study with Table 3. In fact, we can notice that the time of generation process corresponds to the time of acquiring input description.

Table 3: Generation process time statistics

| Reading of input data (1) | Code generation (2) | Total model generation time (1+2) |
|---|---|---|
| 11 min. | 2 sec. | 11 min 2 sec. |

We can finally focus on the result of model generation with Table 4. With such sizes we quickly realized that the analysis of the entire model would be impossible. Indeed given that variables are most of the time boolean, reachability graph of the model can have $2^{1349+7713}$ states. This explains that we have exhausted memory resources when we tried to compute the reachability graph of the model.

Table 4: Generated model sizes

| # state | # flow | # events | # observations |
|---|---|---|---|
| 1349 | 7713 | 2268 | 2388 |

### 4.3 Diagnostic rules

For the last phase it is interesting to have a look at the size of models computed during the task of reduction. Figures 5 and 6 show respectively the proportion of events and the proportion of variables in reduced models. We can see for example that the biggest reduced model do not exceed 90 events and 55 variables while the global model contains 2268 events and 9062 variables.

Finally the time required to compute the 2388 sets of minimal cuts of this model takes about 44 minutes (on a Linux laptop equipped with a Core 2 Duo 2.5Ghz and 4Go of RAM).

## 5 CONCLUSION AND FUTURE WORK

This paper has proposed a feedback on an industrial use of models and minimal cuts for diagnosis. We have presented a complete diagnosis chain for avionics systems starting with acquisition of documents describing the system, continuing with the generation of a model of this system, then generating the set of diagnostic rules for each observation, and finishing with the method of correlation of those rules in order to give the final diagnostic results.

The outcome of this methodology is twofold. First, the generation of diagnostic rules from a correct high-level model, ensure us a better coverage of all possible causes and an increased completeness of rules. Second, providing a totally automatic diagnostic process is an important requirement to make the promotion of model-based approaches in industrial contexts.

Another significant point is the particular structure of our models, due to the way we have built them, which allows this solution to work.

In the future we want to address several topics. First we want to improve different steps of the diagnostic chain to improve the quality of the results. More specifically we want to work on the correlation algorithm and define exactly the abstraction and what are the cases for which we obtain exact results. We also want to explore the possibility to not generate diagnostic rules but to make diagnosis directly from models and so use the dynamic capacity of model-based diagnosis. Another topic we are interested in, is the assessment of the quality of the monitoring. In order to improve diagnostic performances we want to look at diagnosability of our systems and optimization of monitoring functions (Kuntz, 2010).

## ACKNOWLEDGEMENTS

Algorithms mentioned in section 3.2 were designed through a collaboration between THALES Avionics and the LABRI.

We want to thank Sébastien Dubois from THALES Avionics, Aymeric Vincent from LABRI, and Tung Tran Tranh from Université Bordeaux 1 for their contributions to the project.

## REFERENCES

(Arnold *et al.*, 1999) A. Arnold, G. Point, A. Griffault, and A. Rauzy. The ALTARICA formalism for describing concurrent systems. *Fundam. Inf.*, 40:109–124, November 1999.

(Bryant, 1986) R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986.

(Console and Dressier, 1999) L. Console and O. Dressier. Model-based diagnosis in the real world: lessons learned and challenges remaining. In *Proceedings of the 16th international joint conference on Artificial intelligence - Volume 2*, pages 1393–1400, 1999.

(Faure *et al.*, 1999) P.P. Faure, L. Trave-Massuyès, and H. Poulard. An interval model-based approach for optimal diagnosis tree generation. In *10th International Workshop on Principles of Diagnosis (DX'99)*, pages 78–89, June 1999.

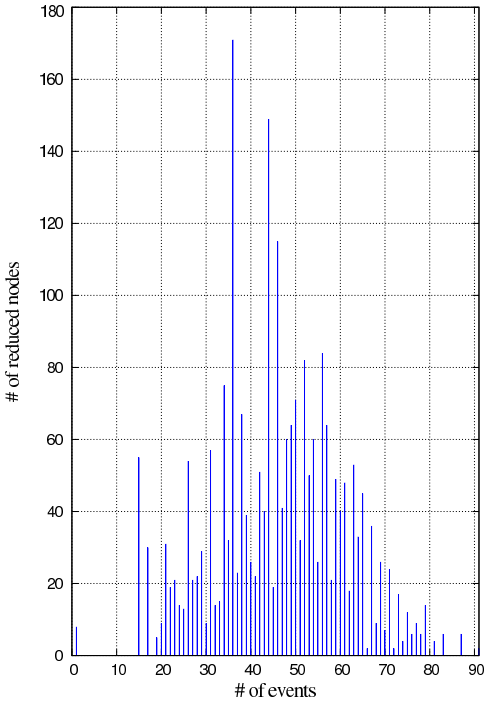


Figure 5: Events in reduced models

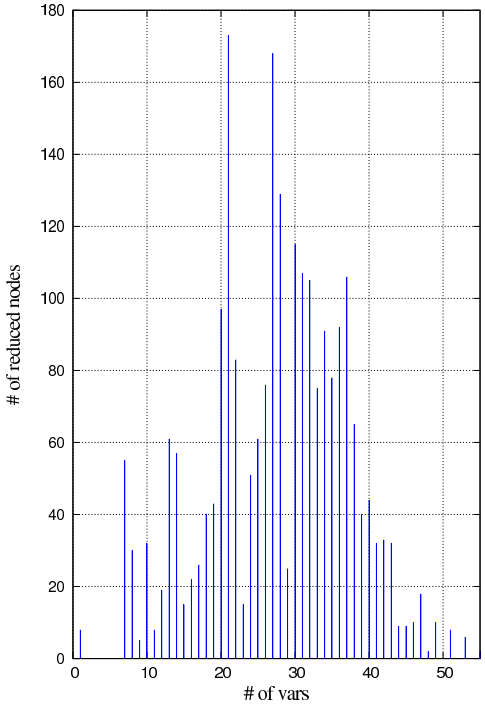


Figure 6: Variables in reduced models

(Griffault and Vincent, 2004) A. Griffault and A. Vincent. The mec 5 model-checker. In *CAV: International Conference on Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, pages 488–491. Springer, July 2004.

(Griffault *et al.*, 2010) A. Griffault, G. Point, and A. Vincent. *AltaRica Checker Handbook*. LaBRI, Talence, France, January 2010. Available on http://altarica.labri.fr/forge/.

(Griffault *et al.*, 2011) A. Griffault, G. Point, and A. Vicent. ALTARICA models and algorithms for diagnosis. Technical report, LaBRI-CNRS, Université de Bordeaux, 2011. Internal collaboration report.

(Hélouët *et al.*, 2006) L. Hélouët, T. Gazagnaire, and B. Genest. Diagnosis from scenarios. In *proc. of the 8th Int. Workshop on Discrete Events Systems, WODES'06*, pages 307–312, 2006.

(Hurdle *et al.*, 2008) E.E. Hurdle, L.M. Bartlett, and J.D. Andrews. System fault diagnostics using fault tree analysis. *Proceedings of the Institution of Mechanical Engineers Part O Journal of Risk and Reliability*, 221(1):43–55, 2008.

(Kuntz, 2010) F. Kuntz. Optimising monitoring of avionics systems to improve diagnostic performances. 10th Summer School MOVEP, pages 100–106, Aachen (Germany), 2010.

(Milner, 1980) R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

(Point and Rauzy, 1999) G. Point and A. Rauzy. ALTARICA - constraint automata as a description language. *European Journal on Automation*, 1999. Special issue on the *Modelling of Reactive Systems*.

(Ramadge and Wonham, 1989) P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81 –98, jan 1989.

(Rauzy, 2001) A. Rauzy. Mathematical foundation of minimal cutsets. *IEEE Transactions on Reliability*, 50(4):389–396, 2001.

(SAE, 2010) SAE. Guidelines for Development of Civil Aircraft and Systems. Technical report, ARP 4754A, 2010.

(Sampath *et al.*, 1996) M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D.C. Teneketzis. Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology*, 4(2):105–124, March 1996.

(Schumann *et al.*, 2004) A. Schumann, Y. Pencolé, and S. Thiébaux. Diagnosis of discrete-event systems using bdds. 15th International Workshop on Principles of Diagnosis (DX-04), 197-202, 2004.

(Tang and Dugan, 2004) Z. Tang and J.B. Dugan. Minimal cut set/sequence generation for dynamic fault trees. In *Proceedings of the 2004 Annual Symposium Reliability and Maintainability*, pages 207–213. IEEE, 2004.